Pods and Services Reference

Feature Summary and Revision History

Summary Data

Table 1: Summary Data

Applicable Products or Functional Area	SMF
Applicable Platform(s)	SMI
Feature Default Setting	Enabled – Always-on
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 2: Revision History

Revision Details	Release
First introduced.	Pre-2020.02.0

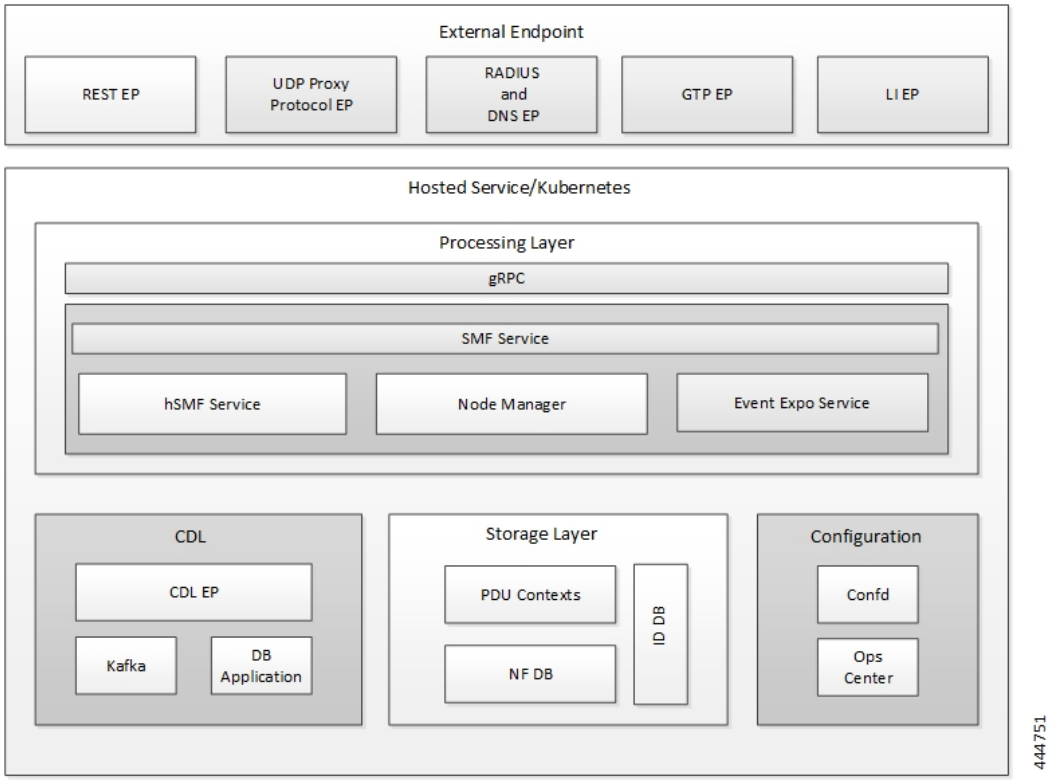
Feature Description

The SMF is built on the Kubernetes cluster strategy, which implies that it has adopted the native concepts of containerization, high availability, scalability, modularity, and ease of deployment. To achieve the benefits offered by Kubernetes, SMF uses the construct that includes the components such as pods and services.

Depending on your deployment environment, the SMF deploys the pods on the virtual machines that you have configured. Pods operate through the services that are responsible for the intrapod communications. If the machine hosting the pods fail or experiences network disruption, the pods are terminated or deleted. However, this situation is transient and SMF spins new pods to replace the invalid pods.

The following workflow provides a high-level visibility into the host machines, and the associated pods and services. It also represents how the pods interact with each other. The representation might defer based on your deployment infrastructure.

Figure 1: Communication Workflow of Pods



Kubernetes deployment includes the kubectl command-line tool to manage the Kubernetes resources in the cluster. You can manage the pods, nodes, and services.

For generic information on the Kubernetes concepts, see the Kubernetes documentation.

For more information on the Kubernetes components in SMF, see the following:

- Pods
- Services

Pods

A pod is a process that runs on your Kubernetes cluster. Pod encapsulates a granular unit known as a container. A pod contains one or multiple containers.

Kubernetes deploys one or multiple pods on a single node which can be a physical or virtual machine. Each pod has a discrete identity with an internal IP address and port space. However, the containers within a pod can share the storage and network resources.

The following tables list the SMF and Common Execution Environment (CEE) pod names and the hosts on which they are deployed depending on the labels that you assign. For information on how to assign the labels, see Associating Pods to the Nodes.

Table 3: SMF Pods

Pod Name	Description	Host Name
api-smf-ops-center	Functions as the *confD* API pod for the SMF Ops Center.	OAM
base-entitlement-smf	Supports Smart Licensing feature.	OAM
cache-pod	Operates as the pod to cache any sort of system information that will be used by other pods as applicable.	Protocol
cdl-ep-session-c1	Provides an interface to the CDL.	Session
cdl-index-session-c1	Preserves the mapping of keys to the session pods.	Session
cdl-slot-session-c1	Operates as the CDL Session pod to store the session data.	Session
documentation	Contains the documentation.	OAM
etcd-smf-etcd-cluster	Hosts the etcd for the SMF application to store information such as pod instances, leader information, NF-UUID, endpoints, and so on.	OAM
grafana-dashboard-app-infra	Contains the default dashboard of app-infra metrics in Grafana.	OAM
grafana-dashboard-cdl	Contains the default dashboard of CDL metrics in Grafana.	OAM
grafana-dashboard-smf	Contains the default dashboard of SMF-service metrics in Grafana.	OAM
gtpc-ep-n0	Operates as GTPC endpoint of SMF.	Protocol
kafka	Hosts the Kafka details for the CDL replication.	Protocol
li-ep-n0	Operates as Lawful Intercept endpoint of SMF.	Protocol
oam-pod	Operates as the pod to facilitate Ops Center actions like show commands, configuration commands, monitor protocol monitor subscriber, and so on.	OAM
ops-center-smf-ops-center	Acts as the SMF Ops Center.	OAM
smart-agent-smf-ops-center	Operates as the utility pod for the SMF Ops Center.	OAM

Pod Name	Description	Host Name
nodemgr-n0	Performs node level interactions such as N4 link establishment, management (heart-beat), and so on. Also, generates unique identifiers such as UE IP address, SEID, CHF-ID, Resource URI, and so on.	Service
protocol-n0	Operates as encoder and decoder of application protocols (PFCP, GTP, RADIUS, and so on) whose underlying transport protocol is UDP.	Protocol
smf-radius-dns-n0	Operates as RADIUS and DNS endpoint of SMF.	Protocol
rest-ep-n0	Operates as REST endpoint of SMF for HTTP2 communication.	Protocol
service-n0	Contains main business logic of SMF.	Service
udp-proxy	Operates as proxy for all UDP messages. Owns UDP client and server functionalities.	Protocol
swift-smf-ops-center	Operates as the utility pod for the SMF Ops Center.	OAM
zookeeper	Assists Kafka for topology management.	OAM

Table 4: CEE Pods

Pod Name	Description	Host Name
alert-logger	Stores the history of active and resolved alerts.	OAM
alertmanager	Duplicates alerts and sends out resolution of alerts when they are resolved in Prometheus.	OAM
api-cee-global-ops-center	Functions as the confD API pod for the CEE Ops Center.	OAM
bulk-stats	Assists to retrieve bulkstats saved by Prometheus containers.	OAM
cee-global-product-documentation	Contains the product documentation (API, CLI, and so on).	OAM
core-retriever	Assists in retrieving the core dumps.	All the nodes except ETCD nodes.
documentation	Contains the documentation (metrics and usage).	OAM
grafana-dashboard-metrics	Assists in collating Grafana metrics on the dashboard.	OAM
grafana	Contains the Grafana metrics for CEE.	OAM
kube-state-metrics	Assists in generating metrics about the state of Kubernetes objects: node status, node capacity (CPU and memory), and so on.	OAM

Pod Name	Description	Host Name
logs-retriever	Assists in retrieving Kernel, Kubelet, and Container level logs through output to JournalD driver.	All the nodes except ETCD nodes.
node-exporter	Exports the node metrics.	All the nodes.
ops-center-cee-global-ops-center	Provides NETCONF and CLI interface to the application.	OAM
path-provisioner	Provisions the local storage volume.	All the nodes except ETCD nodes.
pgpool	*Pgpool* is a middleware that works between *PostgreSQL* servers and a *PostgreSQL* database.	OAM
postgres	Storage of alerts and Grafana dashboards.	OAM
prometheus-hi-res	Stores all metrics and generates alerts by alerting rules.	OAM
prometheus-rules	Contains the default alerting rules and recording rules for Prometheus.	OAM
prometheus-scrapeconfigs-synch	Synchronizes the Prometheus scrape configuration.	OAM
pv-manager	Provisions the local storage volume.	OAM
pv-provisioner	Provisions the local storage volume.	OAM
show-tac-manager	Assists in creating and deleting debug package.	OAM
smart-agent-cee-global-ops-center	Operates as the utility pod for the CEE Ops Center.	OAM
snmp-trapper	Sends the SNMP traps based on triggered alerts.	OAM
swift-cee-global-ops-center	Operates as the utility pod for the CEE Ops Center	OAM
thanos-query-hi-res	Implements the Thanos query for Prometheus HA.	OAM
fluentbit	Assists in log forwarding to the external logs collector.	All the nodes except ETCD nodes.

Services

The SMF configuration is composed of several microservices that run on a set of discrete pods. Microservices are deployed during the SMF deployment. SMF uses these services to enable communication between the pods. When interacting with another pod, the service identifies the pod's IP address to initiate the transaction and acts as an endpoint for the pod.

The following table describes the SMF services and the pod on which they run.

Table 5: SMF Services and Pods

Service Name	Pod Name	Description
base-entitlement-smf	base-entitlement-smf	Supports Smart Licensing feature.
datastore-ep-session	cdl-ep-session-c1	Responsible for the CDL session.
datastore-notification-ep	smf-rest-ep	Responsible for sending the notifications from the CDL to the *smf-service* through *smf-rest-ep*.
datastore-tls-ep-session	cdl-ep-session-c1	Responsible for the secure CDL connection.
documentation	documentation	Responsible for the SMF documents.
etcd	etcd-smf-etcd-cluster-0, etcd-smf-etcd-cluster-1, etcd-smf-etcd-cluster-2	Responsible for pod discovery within the namespace.
etcd-smf-etcd-cluster-0	etcd-smf-etcd-cluster-0	Responsible for synchronization of data among the *etcd* cluster.
etcd-smf-etcd-cluster-1	etcd-smf-etcd-cluster-1	Responsible for synchronization of data among the *etcd* cluster.
etcd-smf-etcd-cluster-2	etcd-smf-etcd-cluster-2	Responsible for synchronization of data among the *etcd* cluster.
grafana-dashboard-app-infra	grafana-dashboard-app-infra	Responsible for the default dashboard of app-infra metrics in Grafana.
grafana-dashboard-cdl	grafana-dashboard-cdl	Responsible for the default dashboard of CDL metrics in Grafana.
grafana-dashboard-smf	grafana-dashboard-smf	Responsible for the default dashboard of SMF-service metrics in Grafana.
gtpc-ep	gtpc-ep-n0	Responsible for inter-pod communication with GTP-C pod.
helm-api-smf-ops-center	api-smf-ops-center	Manages the Ops Center API.
kafka	kafka	Processes the Kafka messages.
li-ep	li-ep-n0	Responsible for lawful-intercept interactions.
local-ldap-proxy-smf-ops-center	ops-center-smf-ops-center	Responsible for leveraging Ops Center credentials by other applications like Grafana.
oam-pod	oam-pod	Responsible to facilitate Exec commands on the Ops Center.

Service Name	Pod Name	Description
ops-center-smf-ops-center	ops-center-smf-ops-center	Manages the SMF Ops Center.
ops-center-smf-ops-center-expose-cli	ops-center-smf-ops-center	To access SMF Ops Center with external IP address.
smart-agent-smf-ops-center	smart-agent-smf-ops-center	Responsible for the SMF Ops Center API.
smf-sbi-service	smf-rest-ep	Responsible for routing incoming HTTP2 messages to REST-EP pods.
smf-n10-service	smf-rest-ep	Responsible for routing incoming N10 messages to REST-EP pods.
smf-n11-service	smf-rest-ep	Responsible for routing incoming N11 messages to REST-EP pods.
smf-n40-service	smf-rest-ep	Responsible for routing incoming N40 messages to REST-EP pods.
smf-n7-service	smf-rest-ep	Responsible for routing incoming N7 messages to REST-EP pods.
smf-nrf-service	smf-rest-ep	Responsible for routing incoming NRF messages to REST-EP pod.
smf-nodemgr	smf-nodemgr	Responsible for inter-pod communication with *smf-nodemgr* pod.
smf-protocol	smf-protocol	Responsible for inter-pod communication with smf-protocol pod
smf-radius-dns	smf-radius-dns	Responsible for inter-pod communication with smf-radius-dns pod
smf-rest-ep	smf-rest-ep	Responsible for inter-pod communication with smf-rest-ep pod
smf-service	smf-service	Responsible for inter-pod communication with smf-service pod
swift-smf-ops-center	swift	Operates as the utility pod for the SMF Ops Center
zookeeper	zookeeper	Assists Kafka for topology management
zookeeper-service	zookeeper	Assists Kafka for topology management

Open Ports and Services

The SMF uses different ports for communication purposes. The following table describes the default open ports and the associated services.

Table 6: Open Ports and Services

Port	Service	Usage
22	SSH	SMI uses this TCP port to communicate with the virtual machines.
80	HTTP	SMI uses this TCP port for providing Web access to CLI, Documentation, and TAC.
443	SSL/HTTP	SMI uses this TCP port for providing Web access to CLI, Documentation, and TAC.
1434	ms-sql-m	SMI uses this UDP port to communicate with BIRD. BIRD is an open source BGP client that is used to exchange routing information between hosts.
6443	HTTP	SMI uses this port to communicate with the Kubernetes API server.
9100	jetdirect	SMI uses this TCP port to communicate with the Node exporter. Node Exporter is a Prometheus exporter for hardware and OS metrics with pluggable metric collectors. It allows you to measure various machine resources such as memory, disk, and CPU utilization.
10250	SSL/HTTP	SMI uses this TCP port to communicate with Kubelet. Kubelet is the lowest level component in Kubernetes. It is responsible for what is running on an individual machine. You can think of it as a process watcher like supervisor but focused on running containers. It has one job: given a set of containers to run, make sure they are all running.
10251		SMI uses this TCP port to interact with the Kube scheduler. Kube scheduler is the default scheduler for Kubernetes and runs as part of the control plane. A scheduler watches for newly created Pods that have no node assigned. For every Pod that the scheduler discovers, the scheduler becomes responsible for finding the best Node for that Pod to run on
10252	apollo-relay	SMI uses this TCP port to interact with the Kube controller. The Kubernetes controller manager is a daemon that embeds the core control loops shipped with Kubernetes. The controller is a control loop that watches the shared state of the cluster through the apiserver and makes changes attempting to move the current state towards the desired state.
10256	HTTP	SMI uses this TCP port to interact with the Kube proxy. Kube proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept. Kube proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.

Converged Core Refactoring Changes

This section describes the changes related to converged core refactoring in this chapter.

SMF Pods

In the converged core architecture, the pod names mentioned in the following table are renamed.

Table 7: Pod Names

Non-converged Core	Converged Core
smf-nodemgr-n0	nodemgr-n0
smf-protocol-n0	protocol-n0
smf-rest-ep-n0	rest-ep-n0
smf-service-n0	service-n0
smf-udp-proxy	udp-proxy

Associating Pods to the Nodes

This section describes how to associate a pod to the node based on their labels.

After you have configured a cluster, you can associate pods to the nodes through labels. This association enables the pods to get deployed on the appropriate node based on the key-value pair.

Labels are required for the pods to identify the nodes where they must get deployed and to run the services. For example, when you configure the protocol-layer label with the required key-value pair, the pods are deployed on the nodes that match the key-value pair.

1. To associate pods to the nodes through the labels, use the following configuration:

```
configure
  label
    cdl-layer
      key key_value
      value value
    oam-layer
      key key_value
      value value
    protocol-layer
      key key_value
      value value
    service-layer
      key key_value
      value value
      end
```

NOTES:

- If you opt not to configure the labels, then SMF assumes the labels with the default key-value pair.
  - **label { cdl-layer { key** *key_value* **| value** *value* **}**: Configures the key value pair for CDL.
  - **oam-layer { key** *key_value* **| value** *value* **}**: Configures the key value pair for OAM layer.
  - **protocol-layer { key** *key_value* **| value** *value* **}**: Configures the key value pair for protocol layer.
  - **service-layer { key** *key_value* **| value** *value* **}**: Configures the key value pair for the service layer.

Viewing the Pod Details and Status

If the service requires additional pods, SMF creates and deploys the pods. You can view the list of pods that are participating in your deployment through the SMF Ops Center.

You can run the kubectl command from the master node to manage the Kubernetes resources.

1. To view the comprehensive pod details, use the following command.

   kubectl get pods -n smf *pod_name* **-o yaml**

The pod details are available in YAML format. The output of this command results in the following information:

- The IP address of the host where the pod is deployed.
- The service and application that is running on the pod.
- The ID and name of the container within the pod.
- The IP address of the pod.
- The current state and phase in which the pod is.
- The start time from which pod is in the current state.

Use the following command to view the summary of the pod details.

kp get pods -n *smf_namespace* **-o wide**

States

Understanding the pod's state lets you determine the current health and prevent the potential risks. The following table describes the pod's states.

Table 8: Pod States

State	Description
Running	The pod is healthy and deployed on a node. It contains one or more containers.
Pending	The application is in the process of creating the container images for the pod.
Succeeded	Indicates that all the containers in the pod are successfully terminated. These pods cannot be restarted.

State	Description
Failed	One ore more containers in the pod have failed the termination process. The failure occurred as the container either exited with non zero status or the system terminated the container.
Unknown	The state of the pod could not be determined. Typically, this could be observed because the node where the pod resides was not reachable.